# ASSIST Demo: A High Level, High Performance, Portable, Structured Parallel Programming Environment at Work[*]

M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, M. Danelutto, P. Pesciullesi, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo

Dept. of Computer Science – University of Pisa – Viale Buonarroti 2, 56127 Pisa

**Abstract.** This work summarizes the possibilities offered by parallel programming environment ASSIST by outlining some of the features that will be demonstrated at the conference *demo session*. We'll substantially show how this environment can be deployed on a Linux workstation network/cluster, how applications can be compiled and run using ASSIST and eventually, we'll discuss some ASSIST scalability and performance features. We'll also outline how the ASSIST environment can be used to target GRID architectures.

**Keywords.** Structured parallel programming, skeletons, coordination languages.

## 1 Demo Background

ASSIST (*A Software development System based on Integrated Skeleton Technology*) is a parallel programming environment based on skeleton and coordination language technology [8,9,3,2]. ASSIST provides the user/programmers with a structured parallel programming language (ASSISTcl), an integrated set of compiling tools (`astCC`) and a portable run time (the actual runtime *CLAM*, and the loader/runner `assistrun`). ASSIST is based on both skeleton and coordination languages technology, and comes after some other different experiences of our group related to skeleton based parallel programming [5,4]. It builds on the experience gained in these projects.

The main goals in the design of ASSIST have been: high level programmability, rapid prototyping and suitability for complex multidisciplinary applications; functional and performance portability across a range of different target architectures; software reuse and interoperability. These goals have been achieved by taking a number of design choices and using several different implementation techniques.

[*] This work has been partially supported by the National Research Council Coordinated Project CNRC0014B3 "Development environment for multiplatform and multilanguage high-performance applications, based upon the objects model and structured parallel programming (Agenzia2000)" and ASI-PQE2000 "Earth observation application development with High Performance Computing Tools" projects

**Programmability:** the coordination language allows programmers to express both standard skeleton/coordination patterns as well as more complex parallelism exploitation patterns. This because a new[1], general purpose, highly configurable parallelism exploitation pattern has been included in the language (the `parmod` one) and because completely general graphs of `parmod`s can be used to describe the parallel structure of applications. Furthermore, ASSISTcl allows a controlled usage of *external objects* (e.g. existing, possibly parallel, libraries) after programmer requests

**Performance:** ASSIST environment design is highly layered. The source code is first translated into an intermediate "task code". The task code, in turn, is compiled to an abstract machine built on top of ACE (*Adaptive Coordination Environment* [1]) and AssistLib. ACE provides communication and process framework. AssistLib is a C++ library implementing specific mechanisms needed to implement task code on top of ACE. The whole compiler design is based on OO design pattern technology, thus allowing easily replacement of compiler parts as well as introduction of new features without affecting the ASSIST support overall design. In addition, the whole compile process and both the task code and the AssistLib level have been carefully optimized in order to avoid any kind of bottlenecks as well as of overhead sources.

**Interoperability and software reuse:** all the sequential portions of code needed to instantiate `parmod` parallelism exploitation patterns within an ASSISTcl program can be written in any of the C, C++ or F77 languages, and the details a programmer has usually to deal with when using such a programming language mix are handled by the ASSISTcl compiling tools (and further languages, such as Java, are being taken into account). Furthermore, sequential portions of code can invoke external CORBA object services, and a whole ASSISTcl application could be automatically wrapped into a CORBA object (IDL code generation is automatic), in such a way that its "parallel services" could be invoked from outside the ASSIST world. Existing, highly optimized, parallel scientific libraries can be integrated into ASSISTcl applications in such a way that they look like "normal" ASSISTcl `parmods` [7]

Our group is currently working to ASSIST enhancement and evolution within several National research projects. Within a couple of National Research Council Strategy projects the ASSIST environment is currently being migrated on GRIDs [6]. Further ASSIST environment enhancements are planned within another, three year, large, National Research Council Project (*GRID.it*). The interested reader can refer to different papers describing ASSIST: [8,9,2,7,6,3].

## 2 ASSIST Framework Setup

ASSIST currently runs on cluster or networks of workstations. In order to install ASSIST the user needs to install some separate, public domain software packages. In particular, ACE, *the Adaptive Communication Environment* is needed, as well

[1] with respect to previously developed skeleton based programming environments

as DVSA, the *Distributed Virtual Shared Areas library.* The former providing the basic functions of ASSIST abstract machine, the latter providing the support for data sharing across network connected processing elements. Both libraries assume a POSIX TCP/IP framework. ACE runs on both Linux and Windows or Mac (OS/X) boxes. DVSA originally runs on top of Linux, and is currently begin tested on the other environments by our research group.

Once these libraries have been configured, the installation of ASSIST is a matter of a couple of `make` commands. The correct installation of the ASSIST package provides user with the `astCC` and `assistrun` commands, i.e. the compiler and run commands, respectively.

*Compiling* ASSIST *programs.* Once an ASSIST program has been produced, using any available editor, it can be compiled using the `astCC` command. The compiler basically produces a set of C++ "object code" files, a kind of configuration file storing an XML representation of the resources needed to run the program, the `makefiles` needed to compile actual object code out of these sources, and eventually executes a sort of `make all` completing the generation of actual object code. Several parameters of the `astCC` command allow, for instance, the source files to be kept after the production of object code, in such a way the programmer may intervene directly at the "task code" or AssistLib level, i.e. at the level of the intermediate abstract machine, if needed.

*Running* ASSIST *programs.* In order to run an ASSIST executable, the user must basically perform two steps: first, a *CLAM* (*Coordination Language Abstract Machine*, the one built out of ACE and AssistLib, basically) instance must be run on the processing elements of the target architecture. This can be done running by hand the *CLAM* process onto every node of the target architecture. ASSIST provides scripts that make this process automatic, once the (IP) names of the nodes are known. This step can be performed once and forall, as *CLAM* is an execution server and it can be invoked multiple times, with different object codes. Every time, *CLAM* loads object code, configuration info and provides to actually run the ASSIST object code. Second, we must issue an `assistrun` command. The command accepts as a parameter the XML configuration file produced by the compiler[2] and consequently properly configures *CLAM* and actually starts computation.

## 3 Programmability

The time needed to develop running, scalable ASSISTcl programs is significantly smaller than the time needed to develop equivalent (both in the functional and in the performance sense) applications with different, more classical parallel programming tools, such as MPI, for instance. The programmer has handy ways to express simple as well as complex parallelism exploitation patterns. All the

[2] and possibly manipulated by the programmer/user

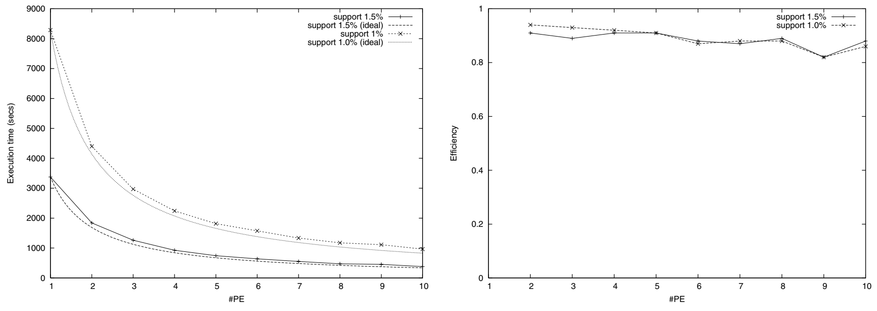

**Fig. 1.** Scalability and efficiency of Apriori data mining application

details needed to implement parallelism exploitation are handled by the compiling tools. Therefore, on the one hand the programmers may write the parallel structure of the application very quickly, while on the other hand performance exploitation is in charge of the tools, and again this consistently shortens the application development time.

## 4 Performance Results

We experimented different synthetic benchmarks as well as complete applications written in ASSISTcl. Typical performance numbers got out of Intel/Linux cluster are depicted in Figure 1 (left). Provided that computational grain is medium to coarse grain, ASSIST demonstrates good speedup and efficiency. The Figure plots values achieved running a data mining application exploiting an "a priori" algorithm.[3] In the Apriori code execution, efficiency is constantly more than 80%, as shown in the Figure 1 (right).

## 5 Interoperability

ASSIST programs can invoke external services/code using several mechanism including typical CORBA ones. In particular, any portion of code included in an ASSIST program may call external CORBA objects methods and a whole ASSISTcl program can be wrapped into a CORBA object whose services/methods[4] can be called from elsewhere. To demonstrate this feature, we prepared a Nbody program whose computational intensive part is performed in an ASSISTcl program but visualizing results accessing an X display via CORBA. The graphical output of the program is in Figure 2. The Nbody program is actually a code implementing the naive, $n^2$ algorithm, as the goal there was only to demonstrate

[3] The parameters of these runs can be summarized as follows: #DBtransactions = 1236000, average transaction len = 30, #items = 1000. Large item-sets: #patterns = 2000, average pattern len = 10, correlation between consecutive patterns = 0.5, average confidence in a rule = 0.75, variation in the confidence = 0.1

[4] computation of the parallel program onto a given input data set, actually

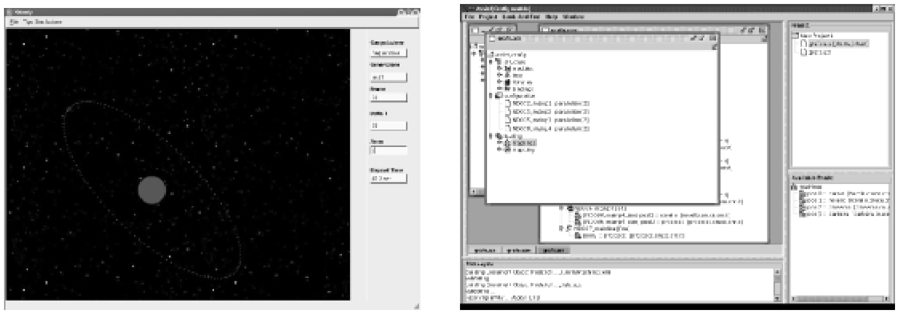

**Fig. 2.** N-body ASSISTcl program interacting with graphic display via CORBA (left) and assistConf configuration tools (right)

interoperability via CORBA, even in case of small granularity ops (i.e. graphic display operations).

## 6 Heterogeneous Target Architecture & GRID

We are currently adapting the ASSISTcl compiling tools to produce object code for heterogeneous architectures (networks). Due to the structured layered implementation of the compiling tools of the ASSIST framework, in order to address heterogeneous architectures we simply have to perform two steps: first, we need to activate the possibility of delivering external data representation messages between hosts. ACE has a full support for such "processor neutral" data representation but at the moment the marshaling and unmarshaling routines do not use such feature. Second, we must arrange the `makefile` production in such a way that different DLLs (object code) are produced for the different machines in the target architectures. When all the needed different object codes are available, the `assistrun` command may exploit them accordingly to the contents of the XML configuration file.

Both these tasks do not present any technical difficulty. They have been postponed only due to lack of human resources (programmers) in the project and we plan to have a working version of ASSISTcl compiler targeting heterogeneous architecture by the end of year 2003.[5]

In the meanwhile, we are moving the whole ASSIST framework to the GRID. As a first step, a tool has been built [6] that allows to manipulate (within a nice graphical interface) the XML configuration file according to information taken from GRID information services, in such a way that ASSISTcl programs can be eventually run on GRIDs. Figure 2 shows a snapshot of the tool. Actually, most of the work needed to run ASSISTcl programs on GRID is to be performed by hand by the programmer. In particular, most of the dynamic features of GRID[6]

[5] development, experiments and debugging is performed on a mixed Linux/Pentium and MacOS/X/PowerPC machines

[6] e.g. resource lookup and reservation

are completely in charge of the programmer interacting with the configuration tool.

## 7 Conclusion

We discussed some features of the ASSIST structured, parallel programming environment, that will be demonstrated during this conference in the new, for Europar, "demo session". More precise information concerning ASSIST can be found in the other papers of our group.

**Acknowledgements.** We wish to thank people that contributed in different, essential ways, to the development of ASSIST: R. Baraglia, D. Laforenza, M. Lettere, D. Guerri, S. Magini, S. Orlando, A. Paternesi, R. Perego, A. Petroccelli, E. Pistoletti, L. Potiti, N. Tonellotto, L. Vaglini, P. Vitale.

## References

1. The Adaptive Communication Environment home page. `http:// www.cs.wustl.edu/ ~schmidt/ACE-papers.html`, 2003.
2. M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, M. Danelutto, P. Pesciullesi, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo. A framework for experimenting with structured parallel programming environment design. In *Proceedings of PARCO'03*, 2003. to appear.
3. M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, S. magini, P. Pesciullesi, L.Potiti, R. Ravazzolo, M. Torquati, M. Vanneschi, and C. Zoccolo. The implementation of ASSIST, an Environment for Parallel and Distributed Programmind. In *Proceedings of Europar'03*, 2003. to appear.
4. B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti, and M. Vanneschi. $P^3L$: A Structured High level programming language and its structured support. *Concurrency Practice and Experience*, 7(3):225–255, May 1995.
5. B. Bacci, M. Danelutto, S. Pelagatti, and M. Vanneschi. SkIE: a heterogeneous environment for HPC applications. *Parallel Computing*, 25:1827–1852, Dec. 1999.
6. R. Baraglia, M. Danelutto, D. Laforenza, S. Orlando, P. Palmerini, R. Perego, P. Pesciullesi, and M. Vanneschi. AssistConf: A Grid Configuration Tool for the ASSIST Parallel Programming Environment. In *Proceedings of the Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 193–200. IEEE, February 2003. ISBN 0-7695-1875-3.
7. P. D'Ambra, M. Danelutto, D. di Serafino, and M. Lapegna. Integrating MPI-Based Numerical Software into an Advanced Parallel Computing Environment. In *Proceedings of the Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 283–291. IEEE, February 2003. ISBN 0-7695-1875-3.
8. M. Vanneschi. ASSIST: an environment for parallel and distributed portable applications. Technical Report TR 02/07, Dept. Comp. Sc., Univ. of Pisa, May 2002.
9. M. Vanneschi. The programming model of ASSIST, an environment for parallel and distributed portable applications. *Parallel Computing*, 28(12):1709–1732, Dec. 2002.